Army Research Laboratory

An Automated Method for Extracting Spatially Varying Time-Dependent Quantities From an ALEGRA Simulation Using VisIt Visualization Software

by Matthew J. Coppinger

ARL-TN-0617 July 2014

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5069

ARL-TN-0617 **July 2014**

An Automated Method for Extracting Spatially Varying Time-Dependent Quantities From an ALEGRA Simulation Using VisIt Visualization Software

Matthew J. Coppinger
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE						*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**						
1. REPORT DATE *(DD-MM-YYYY)* July 2014		**2. REPORT TYPE** Final				**3. DATES COVERED (From - To)** August 2013–January 2014
4. TITLE AND SUBTITLE An Automated Method for Extracting Spatially Varying Time-Dependent Quantities From an ALEGRA Simulation Using VisIt Visualization Software						**5a. CONTRACT NUMBER**
						5b. GRANT NUMBER
						5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Matthew J. Coppinger						**5d. PROJECT NUMBER**
						5e. TASK NUMBER
						5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: RDRL-WMP-A Aberdeen Proving Ground, MD 21005-5069						**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-0617
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)						**10. SPONSOR/MONITOR'S ACRONYM(S)**
						11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT Analysis of the various transient quantities contained in an ALEGRA simulation, including material properties, scalar variables, and tensor variables, can be an arduous task. In particular, the extraction of specific time-dependent quantities traveling through an Eulerian mesh is time consuming. This report documents an automated method of extracting transient quantities that vary spatially from an ALEGRA simulation using a VisIt macro written in the Python programming language. Plots of data extracted using this method are presented.						
15. SUBJECT TERMS modeling, simulation, VisIt, visualization software, depth of penetration, DOP, data extraction						
16. SECURITY CLASSIFICATION OF:				**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Matthew J. Coppinger
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified		UU	22	**19b. TELEPHONE NUMBER** (*Include area code*) 410-278-0815

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

Acknowledgments

The author gratefully acknowledges technical support from John Niederhaus and other members of the ALEGRA development team. The author would also like to thank Drs. W. Casey Uhlig and Andrew J. Porwitzky (Applied Physics Branch, Weapons and Materials Research Directorate) for meaningful discussions.

INTENTIONALLY LEFT BLANK.

1. Introduction

The computational shock multiphysics code ALEGRA is capable of performing multidimensional numerical simulations within an arbitrary Lagrangian-Eulerian framework.[1] ALEGRA simulations produce an abundance of information on various transient quantities, including material properties, scalar variables, and tensor variables. The analysis of specific time-dependent quantities, however, can be difficult. This problem is compounded when the measurement location of the quantity of interest is not fixed. Often, large ALEGRA simulations require massively parallel processing. When massively parallel processing is employed, specialized methods of assembling and then extracting information from an output database are required for post-processing. Utility scripts, such as the specialized history viewer (SHIV), can be used to analyze the transient data of a material characteristic, a global variable, or a particular tracer location via a HISPLT database[2]; however, monitoring particular tracer locations or global material characteristics in many cases is not sufficient to capture the transient quantities of a material traveling through an Eulerian mesh. Visualization software such as VisIt presents an alternative method to examine data through the use of EXODUS databases.[3] In addition, VisIt visualization software allows the user to write macros that can assist in automating the analysis of complex data. This report details an automated method of extracting transient quantities that vary spatially from an EXODUS database using a VisIt macro written in the Python programming language.

2. Macro Description and Implementation

The impetus for creating the macro described in this report came from a repeated analysis of the simulated depth of penetration (DOP) of a shaped charge jet (SCJ). Accordingly, the macro considered here will be based on extracting the location and temperature of the tip of an SCJ as it approaches and penetrates rolled homogenous armor (RHA). The macro was written for VisIt version 2.5.0, and the simulation that was analyzed used a two-dimensional (2-D) axisymmetric geometry with eight elements per millimeter. A 2-D geometry simplifies the data extraction somewhat, but with a small amount of effort, the technique described here can easily be extended to more complicated geometries.

[1] Robinson, A. C.; Carroll, S. K.; Drake, R. R.; Hensinger, D. M.; Labreche, D. A.; Love, E.; Luchini, C. B.; Mosso, S. J.; Niederhaus, J. H. J.; Petney, S. V.; Rider, W. J.; Strack, E.; Weirs, V. G.; Wong, M. K.; Voth, T. E.; Ober, C. C.; Haill, T. A. *ALEGRA User Manual: Version 5.0.*; SAND2010-4796; Sandia National Laboratory: Albuquerque, NM, 2010.

[2] Thompson, S. L.; Kmetyk, L. N. *HISPLT, A Time-History Graphics*; Sandia National Laboratories: Albuquerque, NM, September 1991. Revised April 1994.

[3] Schoof, L. A.; Yarberry, V. R. *EXODUS II: A Finite Element Data*; Sandia National Laboratories: Albuquerque, NM, 1994.

All of the information extracted by this macro can also be extracted manually by interactively selecting a zone or node of interest using VisIt's pick mode. Building a set of spatially varying data as a function of time using this method quickly becomes tedious because the user must select a different node or zone at each timestep and then record the location and temperature. Additionally, VisIt has built-in functionality that allows a user to query certain values over time.

The query-over-time function is useful for monitoring the value of a variable at a specific cell or examining global maxima and minima over a range of timesteps. For example, the query-over-time function could be used if one were interested in knowing the density of a particular cell at each timestep. Queries over time, however, cannot easily track the location of a particular material and then return information on its variables. Fortunately, VisIt allows user-defined macros that, when properly implemented, can both track the location of a particular material and export the location and temperature. Again, the objective of the Python macro described here was straightforward: record the DOP and temperature of the SCJ tip at each timestep. The flowchart depicted in figure 1 illustrates the desired algorithm. The starting point for the macro was a pseudocolor density plot of the 2-D axisymmetric simulation at the initial timestep.

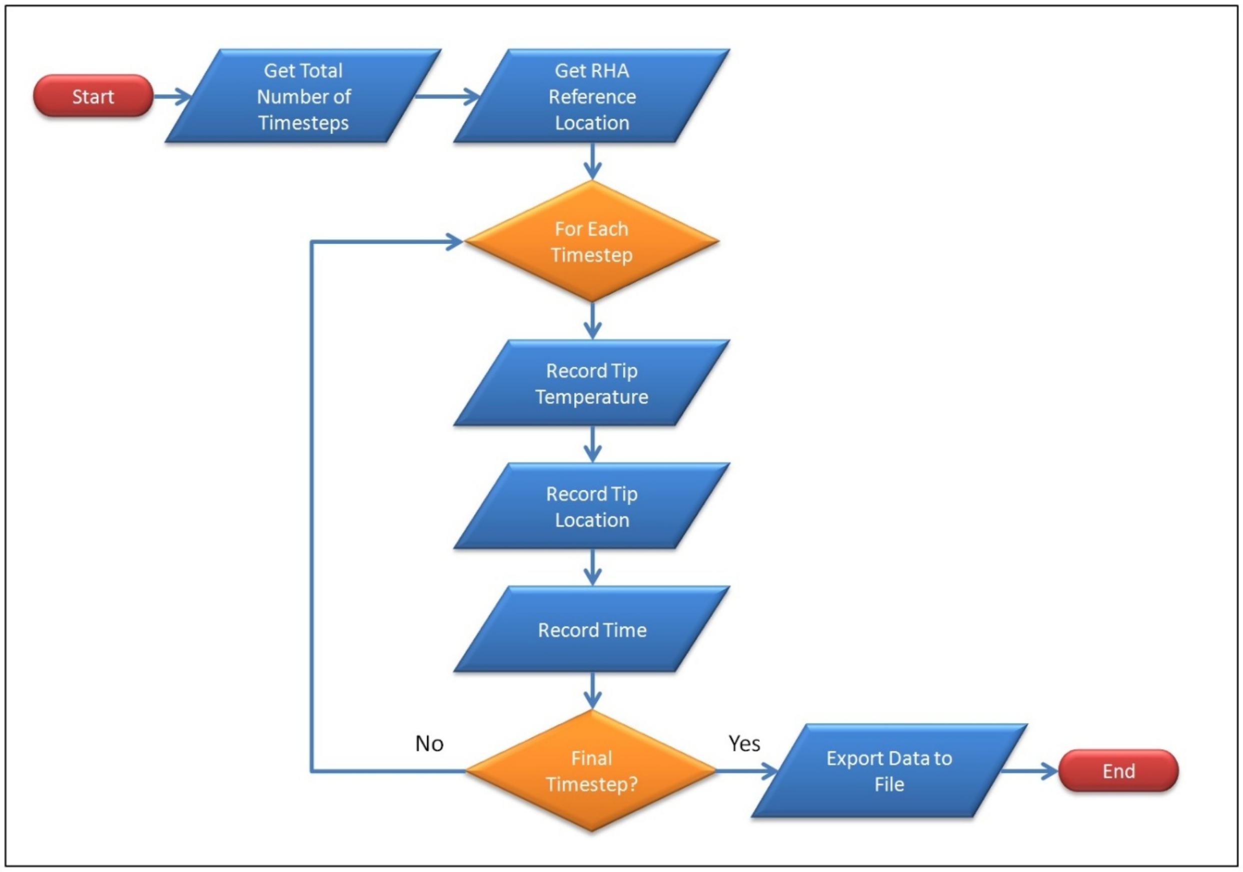

Figure 1. Flowchart depicting the basic algorithm of the macro.

For clarity, the code discussion that follows is broken into subsections. Each subsection gives an overview of a block of the code that performs a simple task. The following line numbers are for discussion only and are not part of the actual script. These are meant to serve as reference points to assist in describing the syntax. To facilitate the full use or modification of the macro by interested users, the macro has been included in its entirety in the appendix.

The first block of code (lines 1–3) determines the number of timesteps in a particular simulation and then outputs the value to the command line interface. Occasionally, complex ALEGRA simulations have been known to crash prior to reaching the ultimate desired length of time. If the total number of intended timesteps is known to the user, the output can serve as a useful quick-check to verify whether or not a simulation ran to completion.

```
1 # Determine the temporal extent of the simulation
2 Total_Timesteps = TimeSliderGetNStates()
3 print "Total number of timesteps = ", Total_Timesteps
```

The second block of code (lines 4–34) performs one of the fundamental tasks in this evaluation: it reads and stores the location of a particular material parameter. In this case the material parameter read is the temperature of the RHA along the axis. The location that is stored as a variable is the front face of the RHA, which is used as a means of identifying the location of the surface of the target. To perform this task, a new window is opened, and the temperature of the RHA (material 7) is plotted (lines 7–11). The maximum and minimum y-coordinates of the simulation are then determined on lines 12 and 13. Because this simulation was axisymmetric, the trajectory of the SCJ was known a priori to be along the y-axis. Once the maximum and minimum coordinates are known, the Lineout function (line 18) is called using the maximum and minimum points along the y-axis as inputs. In an additional window, the Lineout function plots the default variable, i.e., the temperature, along with the coordinates. After these variables have been plotted, the GetPlotInformation function reads the simulation data into user-defined variables so they can be used in calculations or stored to be exported. The code then selects the location of the first nonzero value of the temperature as the front face of the RHA and records the coordinates. This location is used as the origin to measure the DOP. Finally, once the appropriate data has been read, the two plot windows that were opened by the RHA temperature plot and the Lineout function are closed.

```
 4 # Identify a reference location for the measurement.
 5 # In this case, the outer edge of the RHA (material 7)
 6 SetTimeSliderState(0)
 7 AddWindow()
 8 SetActiveWindow(2)
 9 AddPlot("Pseudocolor", "TEMPERATURE_7", 1, 0)
10 ResetView()
11 DrawPlots()
12 View2DAtts = GetView2D()
```

```
13 D = list(View2DAtts.windowCoords)
14 p0 = (0, D[2])
15 p1 = (0, D[3])
16 DefineScalarExpression("x", "coord(Mesh) [0]")
17 DefineScalarExpression("y", "coord(Mesh) [1]")
18 Lineout(p0, p1 , ("default", "x", "y"))
19 SetActiveWindow(3)
20 SetActivePlots(0)
21 Temperature = GetPlotInformation()["Curve"]
22 SetActivePlots(l)
23 x = GetPlotInformation()["Curve"]
24 SetActivePlots(2)
25 y = GetPlotInformation()["Curve"]
26 Temperature_RHA = list(Temperature)

27 ct = 1
28 while Temperature_RHA[ct] == 0:
29    ct = ct+2
30 RHA_start = y[ct]
31 print "RHA starting location =  ", RHA_start
32 DeleteWindow()
33 SetActiveWindow(2)
34 DeleteWindow()
```

The next block of code (lines 35–78) loops through each output timestep in the simulation Exodus file. Similar to the method used in the previous block of code, the temperature variable is used here. The DOP, time, and tip temperatures are all stored as list variables. Again, for the purposes of measuring the DOP, the front face of the RHA is used as a reference. Thus, as illustrated in figure 2, the DOP values are negative until the SCJ tip reaches the RHA. At each timestep, the current tip temperature, DOP, and simulation time are appended to the corresponding list variable.

```
35 # Loop through the meat of the analysis. At each timestep,
36 # calculate the depth of penetration, the temperature, and
37 # record the along with the time
38 DOP_List = range(Total_Timesteps)
39 Time_List = range(Total_Timesteps)
40 Tip_Temp_List = range(Total_Timesteps)
41 for i in range(Total_Timesteps):
42    SetTimeSliderState(i)
43    AddWindow()
44    SetActiveWindow(2)
45    AddPlot("Pseudocolor", "TEMPERATURE_1", 1, 0)
46    ResetView( )
47    DrawPlots()
48    View2DAtts = GetView2D()
```

```
    D = list(View2DAtts.windowCoords)
    pl = (0, D[2])
    p0 = (0, D[3])
    DefineScalarExpression("x", "coord(Mesh)[0]")
    DefineScalarExpression("y", "coord(Mesh)[1]")
    Lineout(p0, p1, ("default", "x", "y"))
    SetActiveWindow(3)
    SetActivePlots(0)
    Temperature = GetPlotInformation () ["Curve"]
    SetActivePlots(l)
    x = GetPlotInformation () ["Curve"]
    SetActivePlots(2)
    y = GetPlotInformation () ["Curve"]
    Temperature_Cu = list(Temperature)
    ct = 1
    while Temperature_Cu[ct] == 0:
        ct = ct+2
    Cu_end = y[ct]
    print "Cu tip ending location =" Cu end
    DeleteWindow( )
    SetActiveWindow(2)
    DeleteWindow( )
    DOP = Cu end - RHA start
    print "Current depth of penetration ", DOP
    DOP_List[i] = DOP
    print "Current tip temperature = ", Temperature_Cu[ct]
    Tip_Temp_List[i] = Temperature_Cu[ct]
    Query("Time")
    Time_List[i] = GetQueryOutputValue()
print "Final depth of penetration = ", DOP, "\n"
```

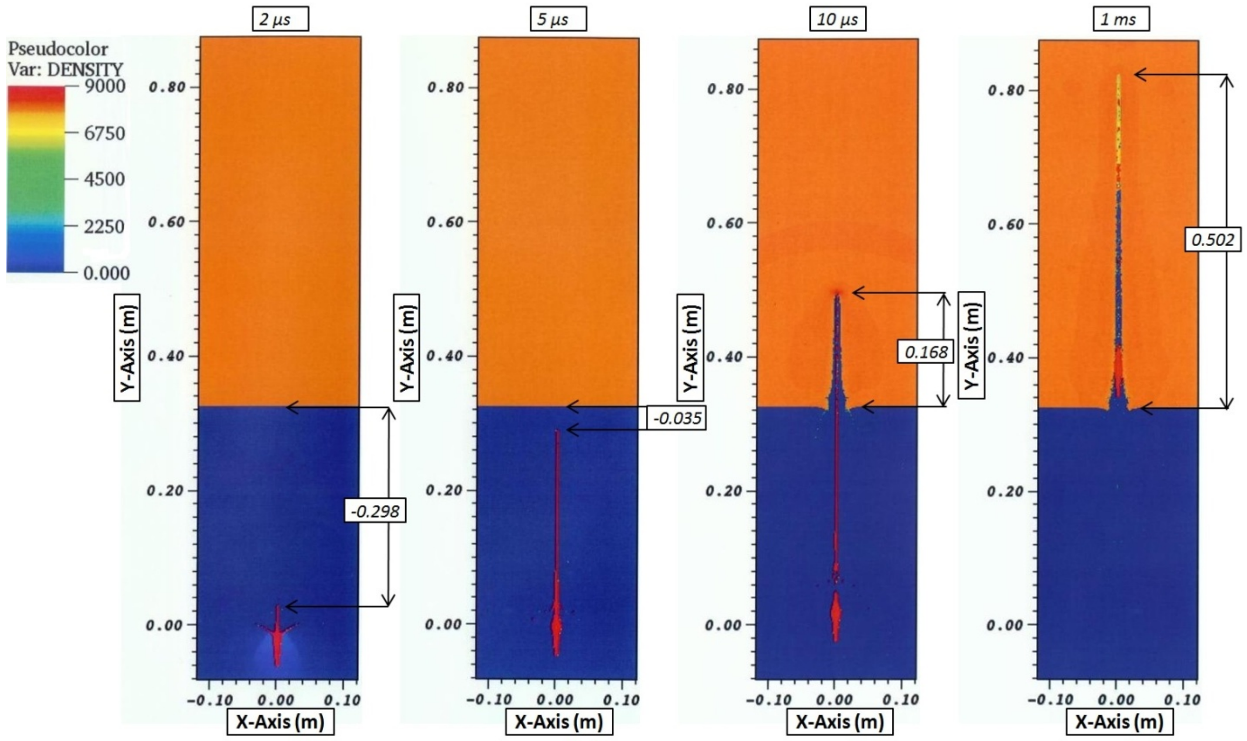

Figure 2. Several snapshots of single frames of the simulation illustrating the sign convention used for the DOP. Until the tip of the SCJ reaches the RHA, the DOP is negative, indicating the distance that remains for the tip to travel prior to reaching the RHA.

The final portion of the code is a method to save the three list variables into a single text file (lines 79–92). This text file can be easily imported into a spreadsheet or graphing software for analysis. In the specified directory, the open function on line 80 will create a new file, filename.txt, if it does not exist or overwrite an existing file with that name. In order to output the variables, each list is converted to a string. When this conversion occurs, brackets are automatically included as characters in the string. To simplify the data processing, prior to writing the data to the text file the macro removes the brackets. (Note: The second argument in the replace field of the string variables on lines 82, 83, 85, 86, 88, and 89 is a single open quote directly followed by a single close quote.)

```
79 # Output the time and the DOP
80 f = open('/home/username/Desktop/filename.txt', 'w')
81 S1 = str(Time_List)
82 S1 = S1.replace('[', '')
83 S1 = S1.replace(']', '')
84 S2 = str(DOP_List)
85 S2 = S2.replace('[', '')
86 S2 = S2.replace(']', '')
87 S3 = str(Tip_Temp_List)
88 S3 = S3. replace('[', '')
89 S3 = S3. replace(']', '')
```

```
output = S1 + "\n" + S2 + "\n" + S3
f.write(output)
f.close ()
```

Once the data have been imported into a spreadsheet or graphing software, they can easily be plotted. Figures 3 and 4 illustrate example plots of the data output by the macro. Figure 3 is a plot of the DOP versus time, and figure 4 is a plot of the temperature versus the SJC tip location relative to the RHA.

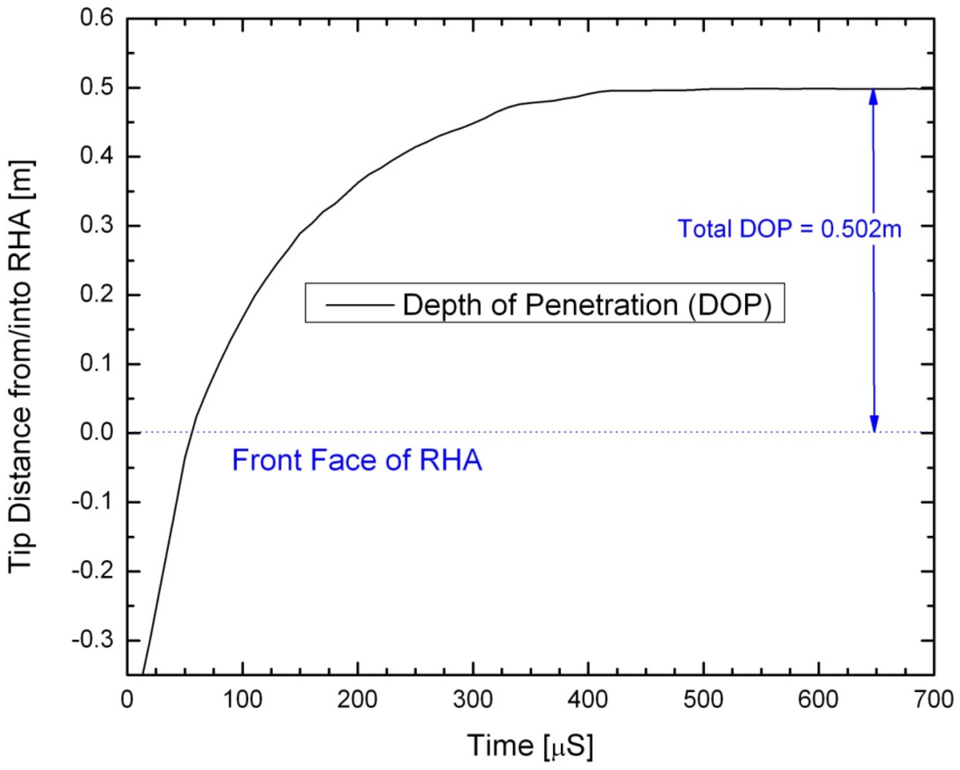

Figure 3. Plot of the DOP versus time for an SCJ as recorded by the VisIt macro. Note that the DOP is negative until the tip of the SCJ reaches the front face of the RHA.

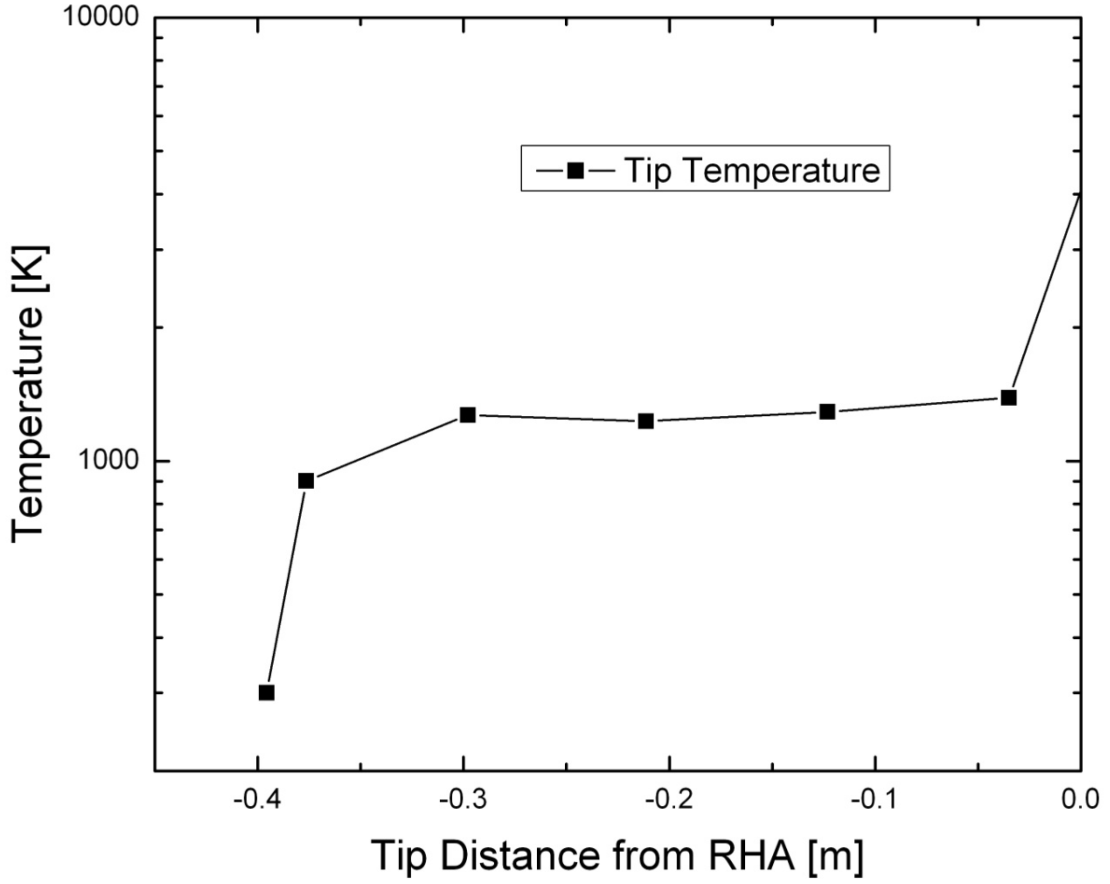

Figure 4. The variation in temperature of the tip of an SCJ with respect to its distance from the front face of the RHA as recorded by the VisIt macro.

3. Conclusion

In summary, this report describes a macro that automates the extraction of data from ALEGRA simulations using the VisIt visualization software. Although the analysis discussed here was for a 2-D axisymmetric simulation, the basic function of the macro can be generalized for any simulation. In addition, automating the extraction of temporally dependent material parameters provides the added benefit of extrapolating the endpoint should the simulation not run to completion. The macro described in this report should provide a valuable starting point for any ALEGRA user who would like to automatically extract temporally and spatially varying data from a simulation.

Appendix. Full Python Macro for Extracting the Depth of Penetration and the Temperature as Functions of Time

This appendix appears in its original form, without editorial change.

```
# Determine the temporal extent of the simulation
Total_Timesteps = TimeSliderGetNStates()
print "Total number of timesteps = ", Total_Timesteps

# Identify a reference location for the measurement.
# In this case, the outer edge of the RHA (material 7)
SetTimeSliderState(0)
AddWindow()
SetActiveWindow(2)
AddPlot("Pseudocolor", "TEMPERATURE_7", 1, 0)
ResetView()
DrawPlots()
View2DAtts = GetView2D()
D = list(View2DAtts.windowCoords)
p0 = (0, D[2])
p1 = (0, D[3])
DefineScalarExpression("x", "coord{Mesh) [0]")
DefineScalarExpression("y", "coord{Mesh) [1]")
Lineout(p0, p1 , ("default", "x", "y"))
SetActiveWindow(3)
SetActivePlots(0)
Temperature = GetPlotInformation()["Curve"]
SetActivePlots(l)
x = GetPlotInformation()["Curve"]
SetActivePlots(2)
y = GetPlotInformation()["Curve"]
Temperature_RHA = list(Temperature)

ct = 1
while Temperature_RHA[ct] == 0:
  ct = ct+2
RHA_start = y[ct]
print "RHA starting location =  ", RHA_start
DeleteWindow()
SetActiveWindow(2)
DeleteWindow()

# Loop through the meat of the analysis. At each timestep,
# calculate the depth of penetration, the temperature, and
# record them along with the time
DOP_List = range(Total_Timesteps)
Time_List = range(Total_Timesteps)
Tip_Temp_List = range(Total_Timesteps)
for i in range(Total_Timesteps):
  SetTimeSliderState(i)
  AddWindow()
```

```
    SetActiveWindow(2)
    AddPlot("Pseudocolor", "TEMPERATURE_1", 1, 0)
    ResetView( )
    DrawPlots()
    View2DAtts = GetView2D()
    D = list(View2DAtts.windowCoords)
    pl = (0, D[2])
    p0 = (0, D[3])
    DefineScalarExpression("x", "coord(Mesh)[0]")
    DefineScalarExpression("y", "coord(Mesh)[1]")
    Lineout(p0, p1, ("default", "x", "y"))
    SetActiveWindow(3)
    SetActivePlots(0)
    Temperature = GetPlotInformation () ["Curve"]
    SetActivePlots(l)
    x = GetPlotInformation () ["Curve"]
    SetActivePlots(2)
    y = GetPlotInformation () ["Curve"]
    Temperature_Cu = list(Temperature)
    ct = 1
    while Temperature_Cu[ct] == 0:
            ct = ct+2
    Cu_end = y[ct]
    print "Cu tip ending location =" Cu end
    DeleteWindow( )
    SetActiveWindow(2)
    DeleteWindow( )
    DOP = Cu end - RHA start
    print "Current depth of penetration ", DOP
    DOP_List[i] = DOP
    print "Current tip temperature = ", Temperature_Cu[ct]
    Tip_Temp_List[i] = Temperature_Cu[ct]
    Query("Time")
    Time_List[i] = GetQueryOutputValue()

print "Final depth of penetration = ", DOP, "\n"
# Output the time and the DOP
f = open('/home/username/Desktop/filename.csv', 'r+')
S1 = str(Time_List)
S1 = S1.replace('[', '')
S1 = S1.replace(']', '')
S2 = str(DOP_List)
S2 = S2.replace('[', '')
S2 = S2.replace(']', '')
S3 = str(Tip_Temp_List)
S3 = S3. replace('[', '')
S3 = S3. replace(']', '')
output = S1 + "\n" + S2 + "\n" + S3
f.write(output)
f.close ()
```

1	DEFENSE TECHNICAL
(PDF)	INFORMATION CTR
	DTIC OCA
2	DIRECTOR
(PDF)	US ARMY RESEARCH LAB
	RDRL CIO LL
	IMAL HRA MAIL & RECORDS MGMT
1	GOVT PRINTG OFC
(PDF)	A MALHOTRA
3	SANDIA NTL LAB
(PDF)	E STRACK
	J NIEDERHAUS
	S PETNEY
36	DIR USARL
(PDF)	RDRL WM
	R DONEY
	W WINNER
	RDRL WMP
	D LYON
	S SCHOENFELD
	RDRL WMP A
	P BERNING
	M COPPINGER
	J FLENIKEN
	C HUMMER
	R MUDD
	A PORWITZKY
	J POWELL
	W UHLIG
	RDRL WMP B
	C HOPPEL
	RDRL WMP C
	R BECKER
	S BILYK
	T BJERKE
	B LEAVY
	S SEGLETES
	C WILLIAMS
	RDRL WMP D
	A BARD
	S HUG
	M KEELE
	D KLEPONIS
	H MEYER
	F MURPHY
	J RUNYEON
	G VUNNI
	S SCHRAML
	M ZELLNER
	RDRL WMP E
	D HACKBARTH
	P BARTKOWSKI
	D HORNBAKER
	J HOUSKAMP
	P SWOBODA
	RDRL WMP F
	N GNIAZDOWSKI
	RDRL WMP G
	N ELDREDGE

INTENTIONALLY LEFT BLANK.